Who is Galen Hunt?

Detours

Library for intercepting arbitrary binary functions:

- used by nearly every MS product team.
- licensed by over 100 third-parties.
- maintained for 15 years

Experiment 19

Prototype for Phone 8

- MinWin, CoreCLR, and Silverlight
- refactored and built for ARM

Drawbridge

Lightweight alternative to hypervisor VMs:

- *picoprocess* sandbox
- refactored Windows as *LibOS*

- And some other stuff… **Windows Media Player** prototype, **BIG**, **Singularity**, etc.
- How do I work? I recruit people smarter than me so I can learn from them …

What is an Engineering System?

- **The end-to-end tools and processes for turning source code into deployed products and services**
  - "I have a line of source code…"
  - "… how do I **submit** it?"
  - "… how do I **build** it?"
  - "… how do I **test** it?"
  - "… how do I **flight** it?
  - "… how do I **deploy** it?"
  - "… how do I **service** it?"
  - "… how do I **refactor** it?
  - "… how do I **include** it in another product?"
  - "… how do I **understand** its use in the practice?"
  - "… how do I **discover** who depends on it?"

What is the Goal of all Engineering Systems?

What is the Goal of all Engineering Systems?

- Support the business model:
  - **The end-to-end tools and processes for turning source code into $$$**
  - **… make it easier for you to produce customer value**

What is the goal of ***One Engineering System***?

- *One Engineering System* is a initiative to create an Engineering System that:
  - enables **reuse** and **sharing** of code across products
  - fosters **collaboration**
  - **removes barriers** and **reduces friction**
  - across **all** of Microsoft

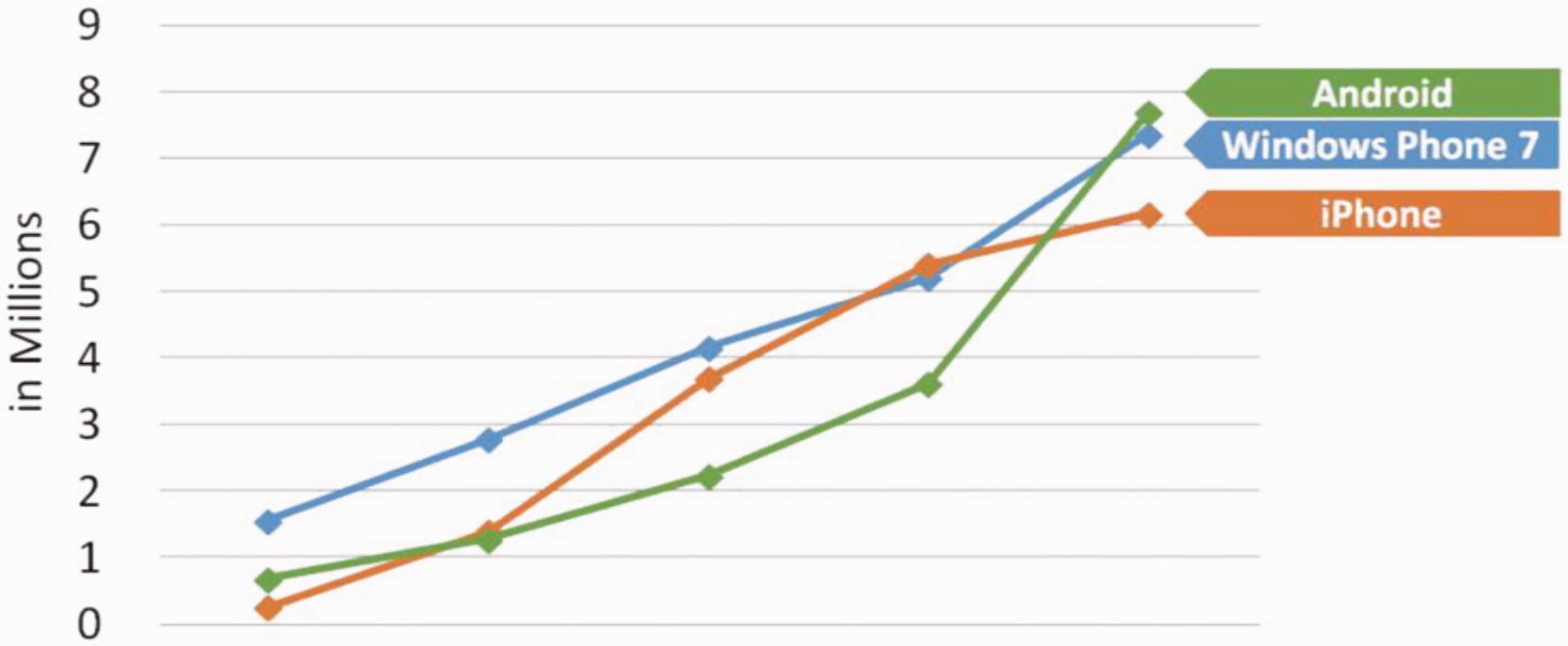
Agenda

- **Motivation:** Why a new Engineering System?
- **Direction:** What will Apex look like?
- **Call to Action:** What should you do now?

a cautionary tale…

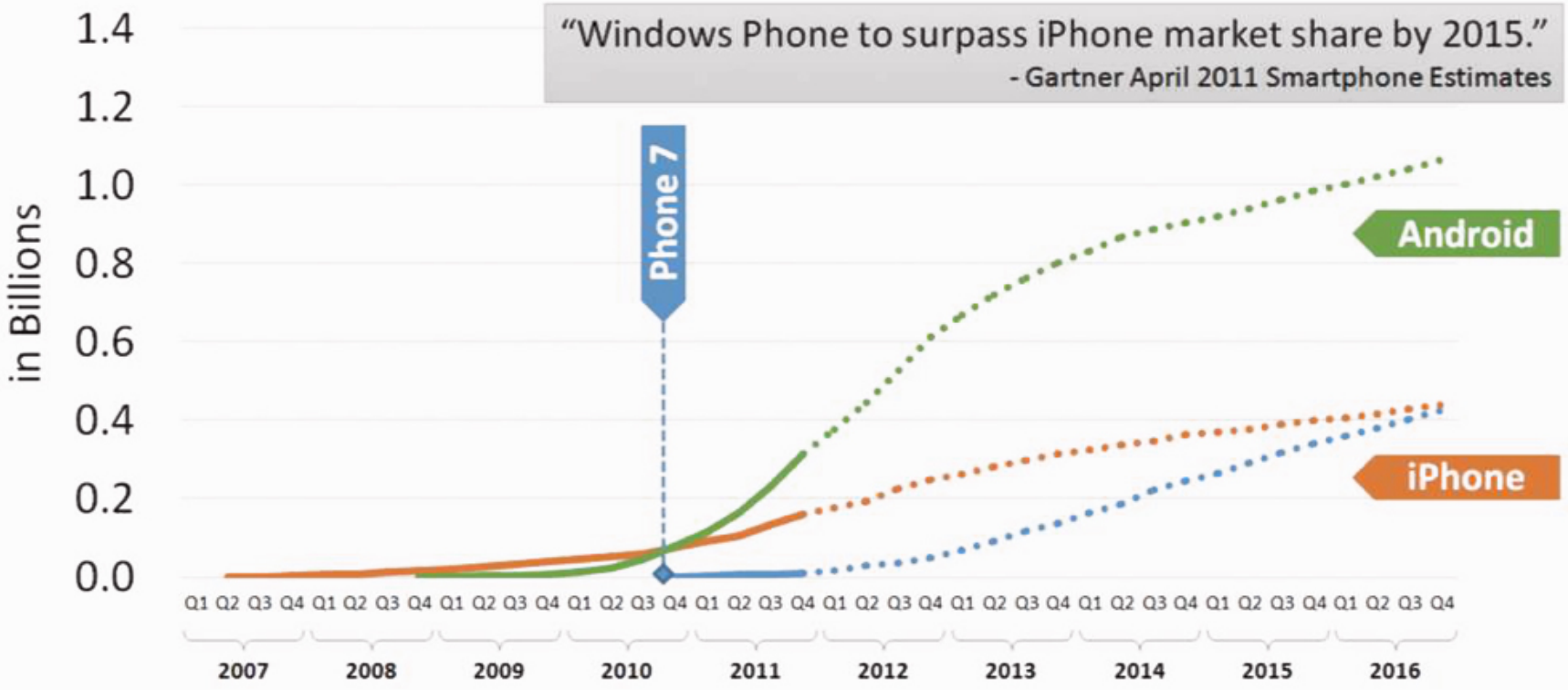
Smartphone Unit Volume

First Year– Plot of cumulative Running Total



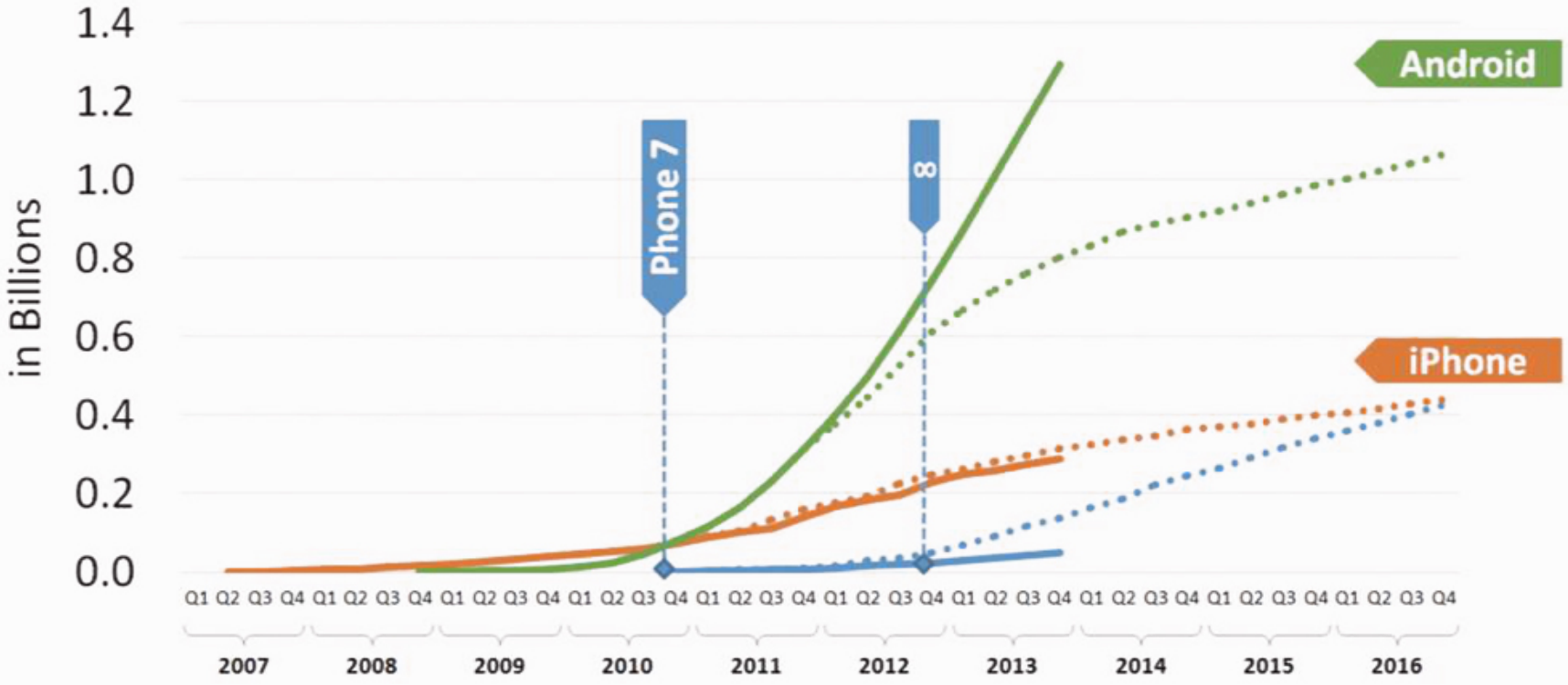
* Data Source: IDC Mobile Tracker 2011

Smartphone Unit Volume Projections (2 y.r.t.)



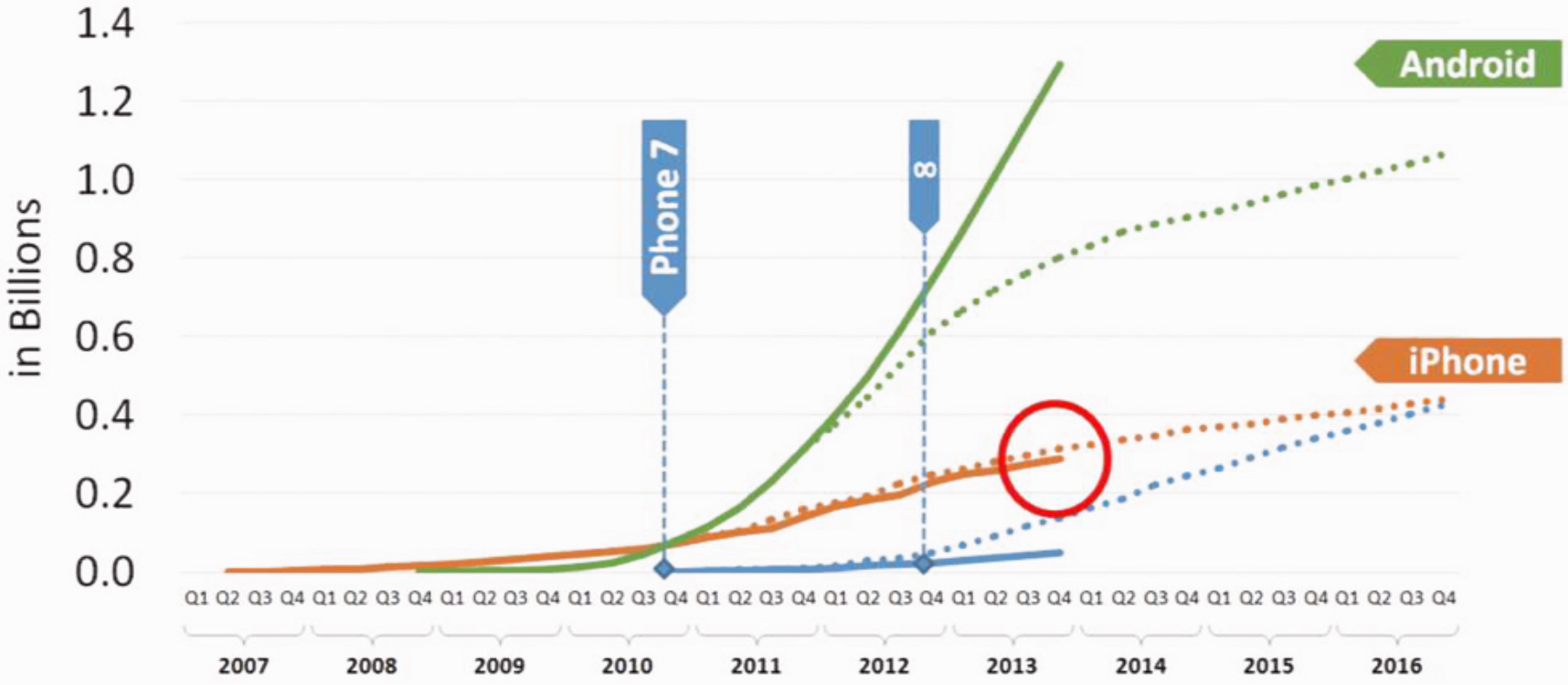
* Data Source: IDC Mobile Tracker 2011

Smartphone Unit Volume through 2013 (2 y.r.t.)



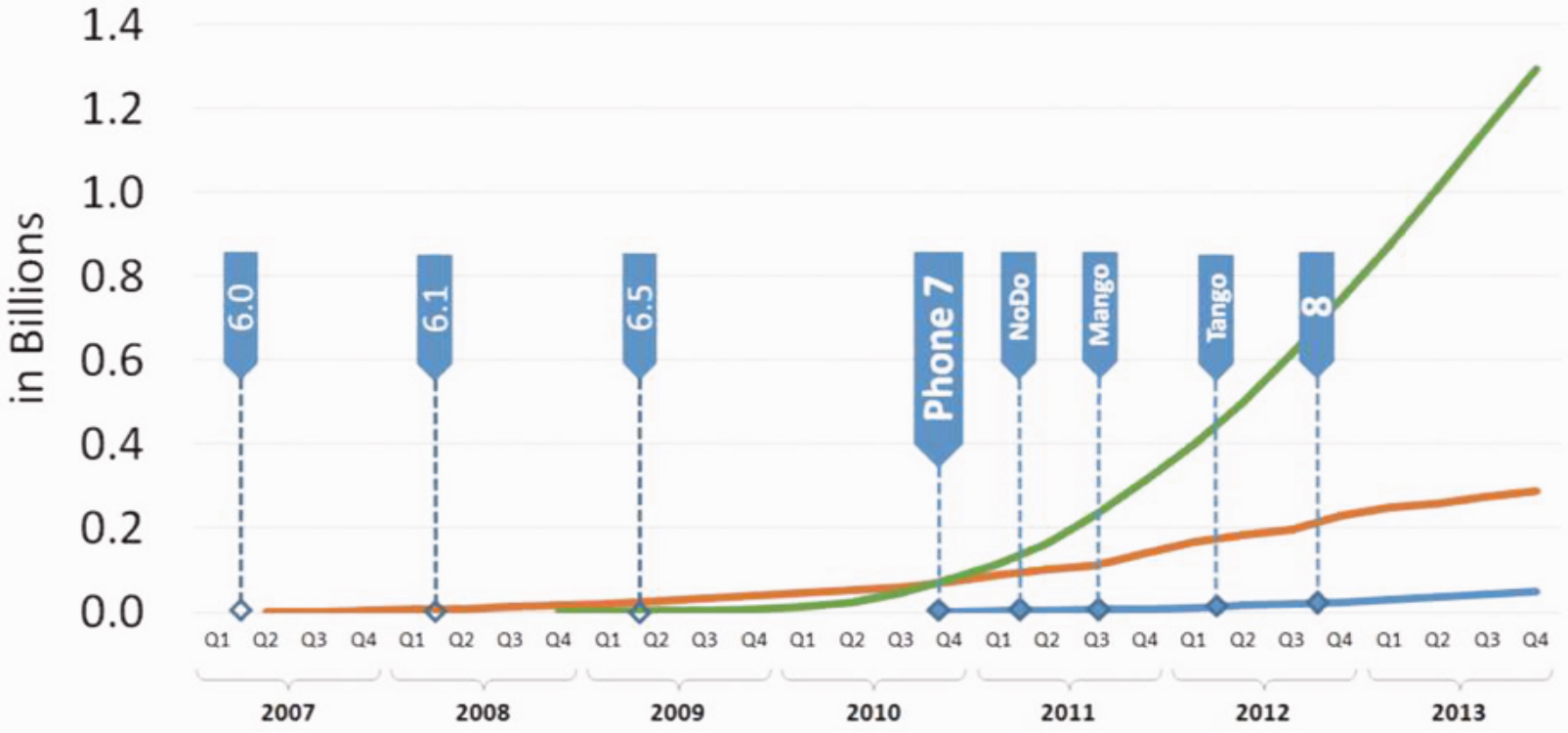
* Data Source: IDC Mobile Tracker 2011 & 2013

Smartphone Unit Volume through 2013 (2 y.r.t.)



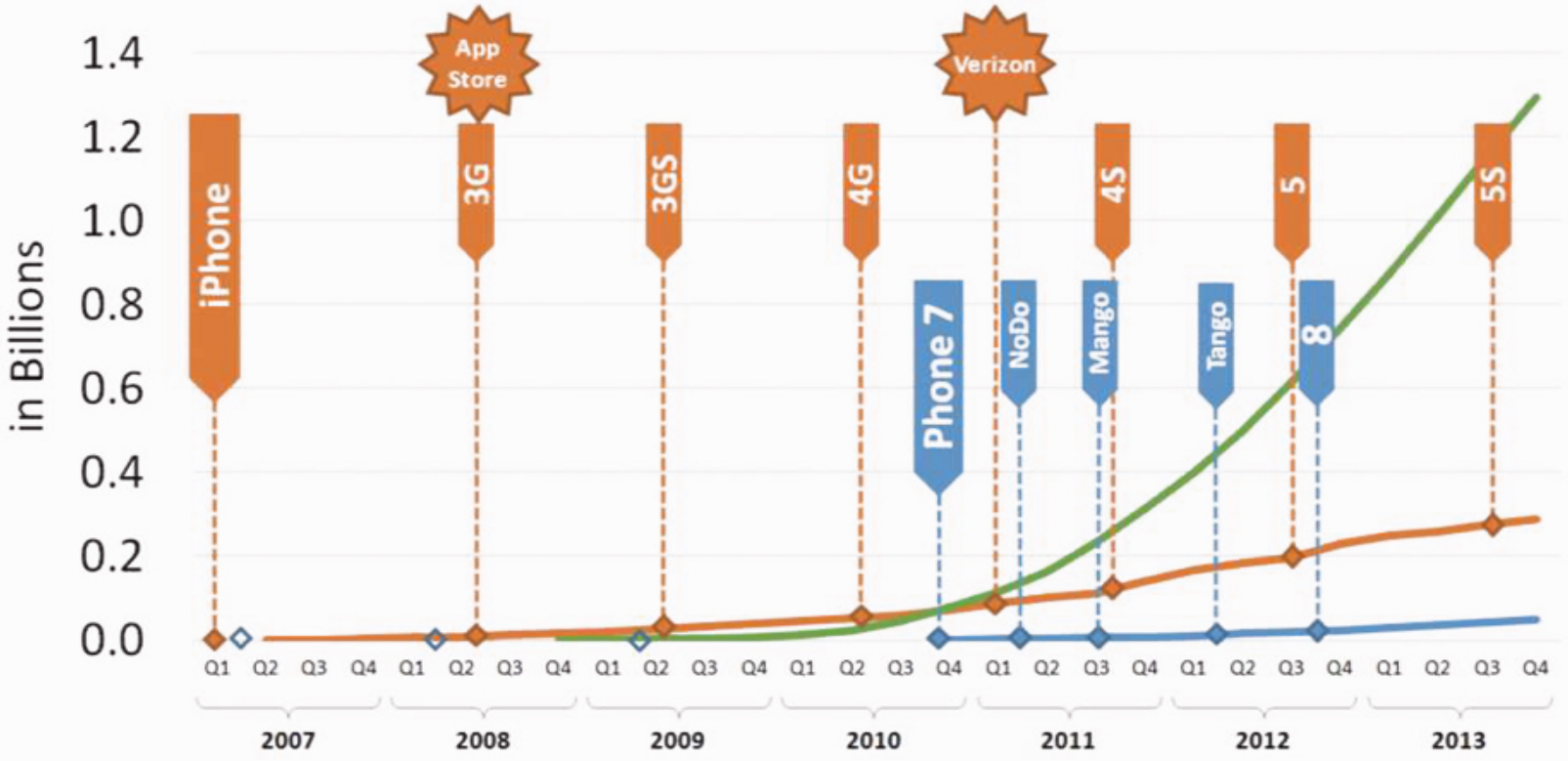
* Data Source: IDC Mobile Tracker 2011 & 2013

Smartphone Unit Volume through 2013 (2 y.r.t.)



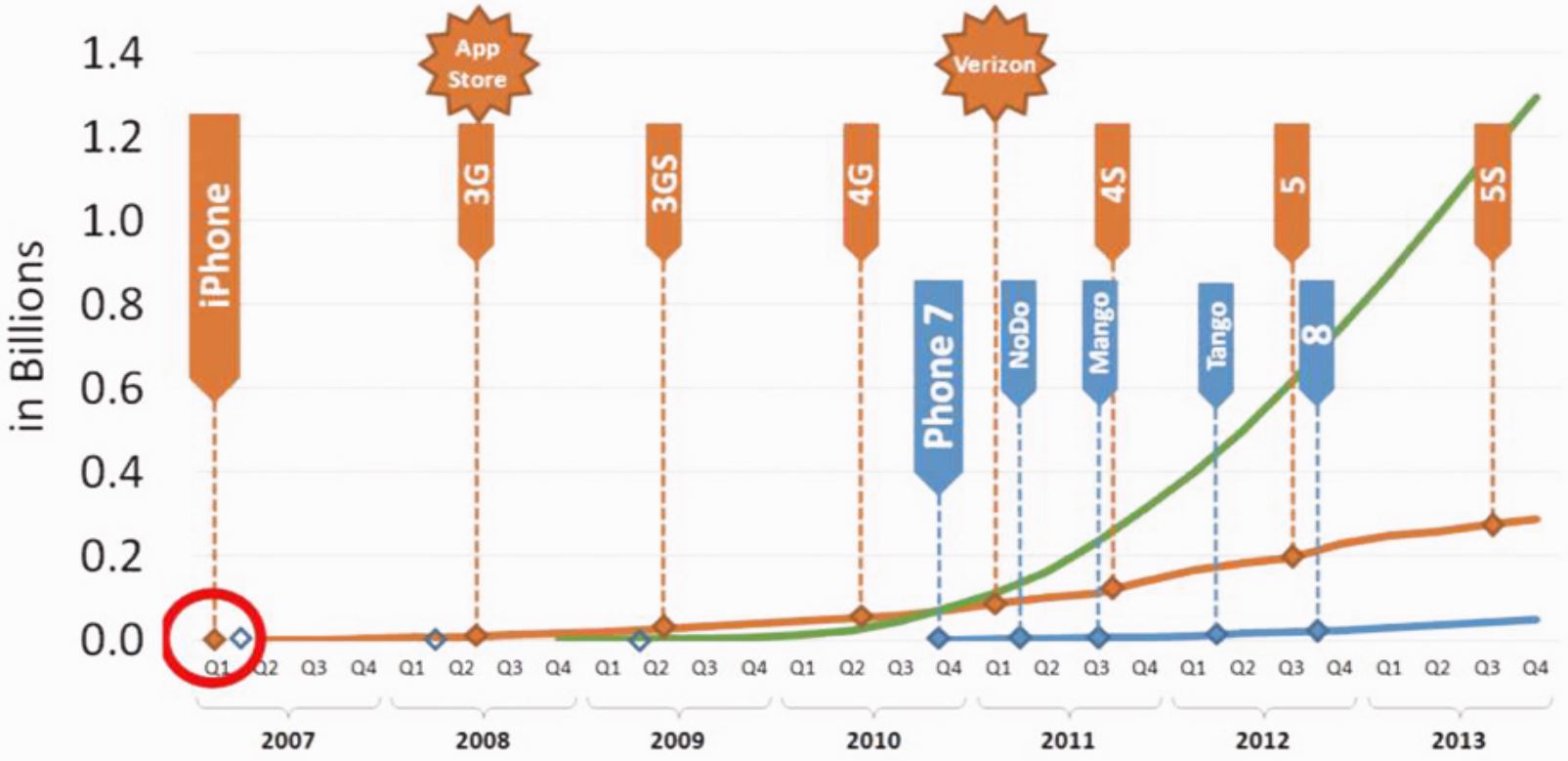
* Data Source: IDC Mobile Tracker 2013

Smartphone Unit Volume through 2013 (2 y.r.t.)



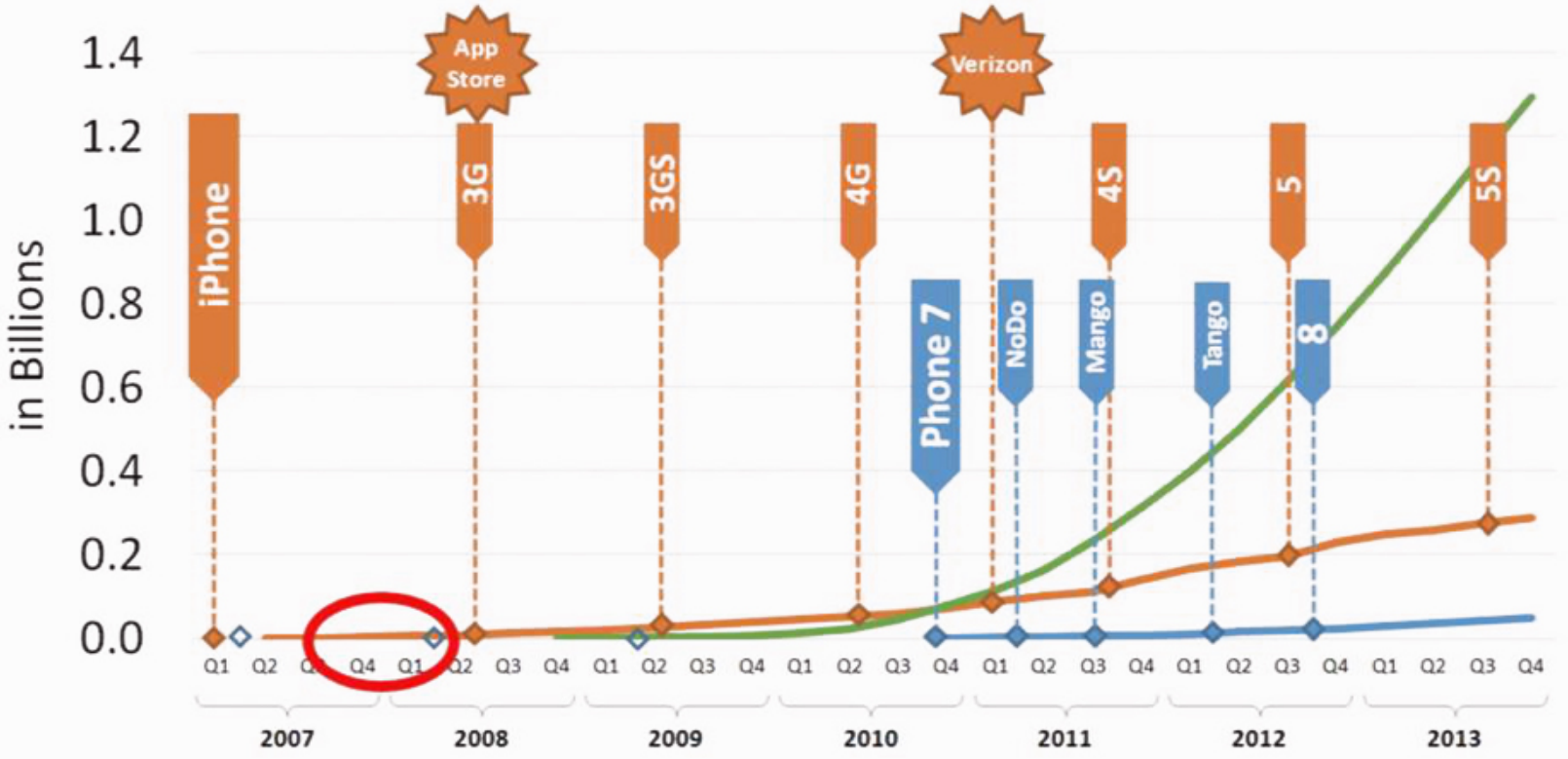
* Data Source: IDC Mobile Tracker 2013

Smartphone Unit Volume through 2013 (2 y.r.t.)



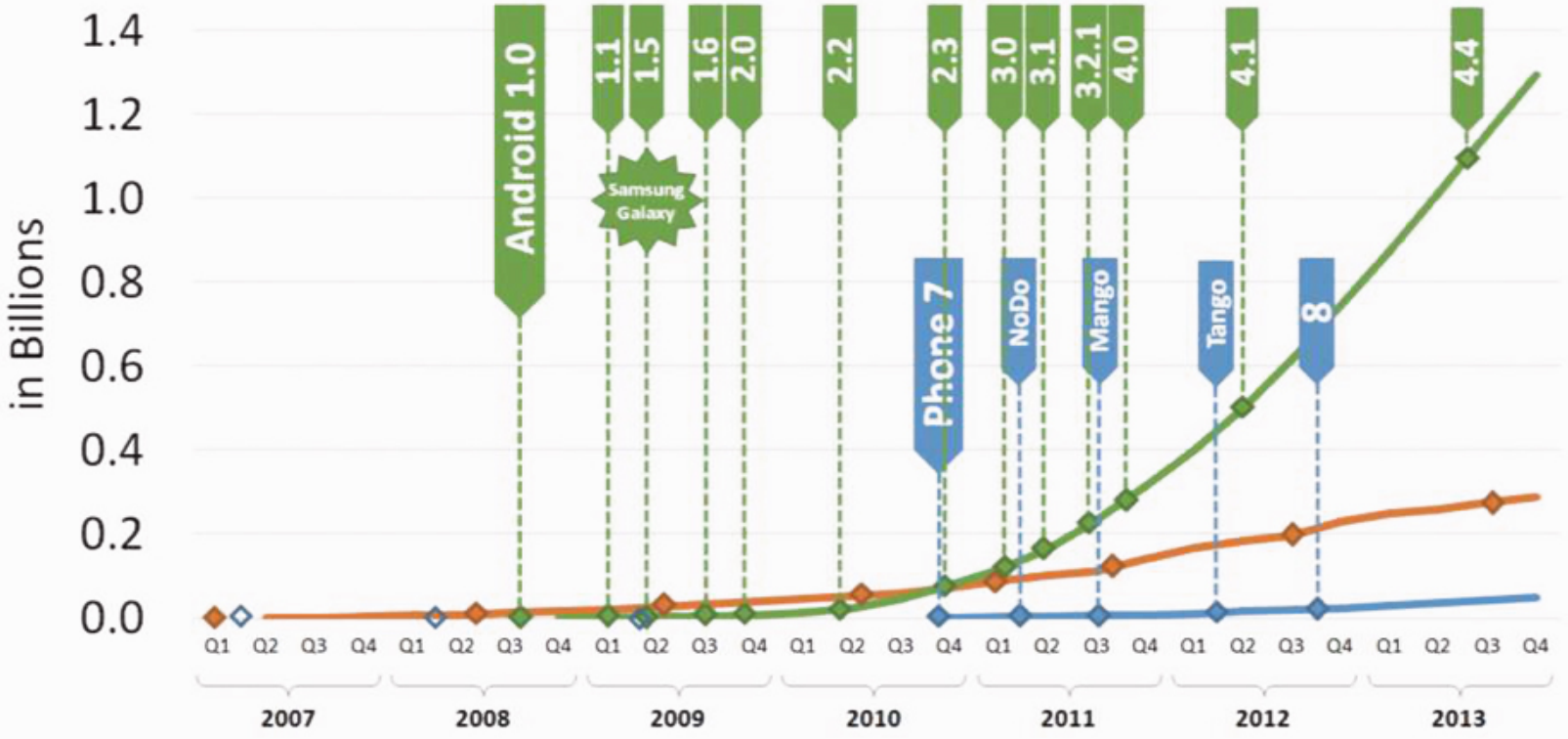
* Data Source: IDC Mobile Tracker 2013

Smartphone Unit Volume through 2013 (2 y.r.t.)



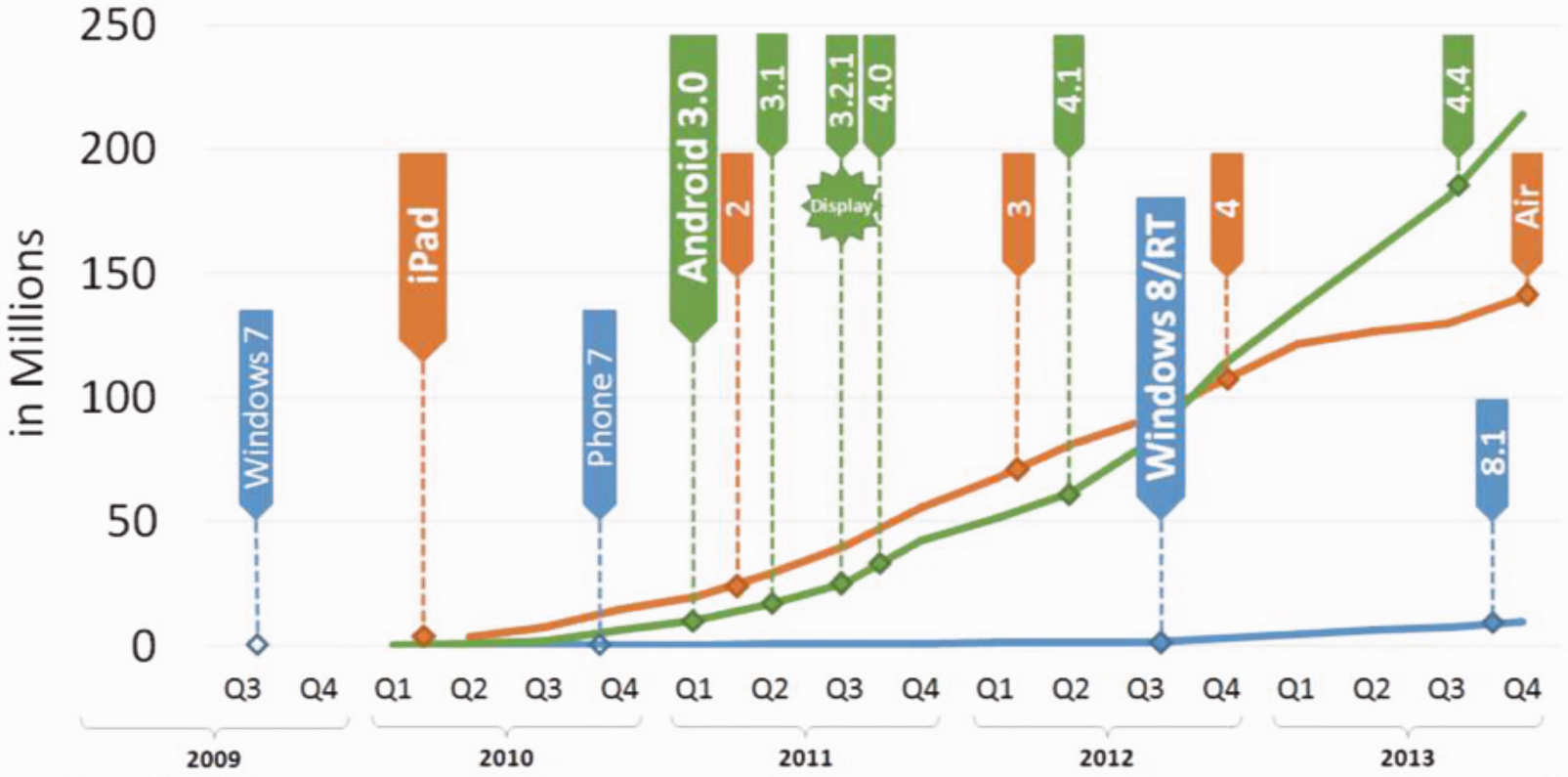
* Data Source: IDC Mobile Tracker 2013

Smartphone Unit Volume through 2013 (2 y.r.t.)



* Data Source: IDC Mobile Tracker 2013

Tablet Unit Volume 2010-2013 (2 y.r.t.)



* Data Source: IDC Tablet Tracker 2013

Other areas in danger…

- **Wireless routers**
- **SmartTVs**
- **Wearables**
- In-flight entertainment
- In-dash navigation, etc.
- Robotics
- **Networked-connected sensors and devices** (IoT)
- **Cloud appliances**

Why aren't we winning?

- ~~enables **reuse** and **sharing** of code across products~~

❑our code is **too entangled** to be shared without forking

- ~~fosters **collaboration**~~

❑our **isolated code islands** prohibit transparent interaction

- ~~**removes barriers** and **reduces friction**~~

❑**tribal knowledge and tenting** prevent collaboration

- ~~across **all** of Microsoft~~

❑we fork when we should merge

❑… we failed to adjust our rate of innovation

Three+ Acid Tests for One Engineering System

- Can I **find** and **enlist** in the sources for any component of any product in the company w/o sending an email?
- Can I **change** a line of code, **build** the component, **verify** that it passes all RI gates, and **flight** it w/o sending an email?
- Can I **archive** my changes in a discoverable branch, **submit** a code review to the maintainers, and expect them to **accept** it?
- … answer the questions again, but this time I incorporate the **serviced** component into my **new** product.

… in an evolving world, with accruing debt,
the most dangerous choice is the status quo.

Agenda

- Motivation: Why a new Engineering System?
- **Direction:** What will Apex look like?
- **Call to Action:** What should you do now?

What is Apex?

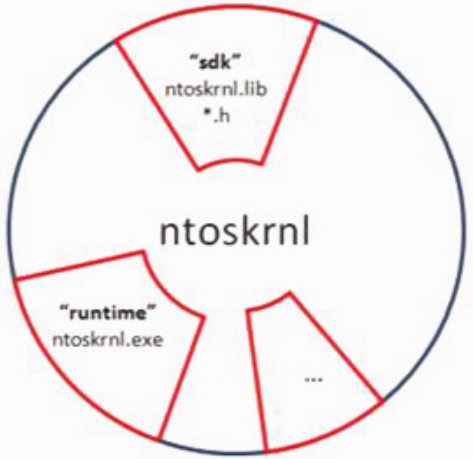
- A **combined effort** between OSG, DevDiv, and MSR
  to incubate One Engineering System
  by merging 1st party and 3rd party tool efforts.

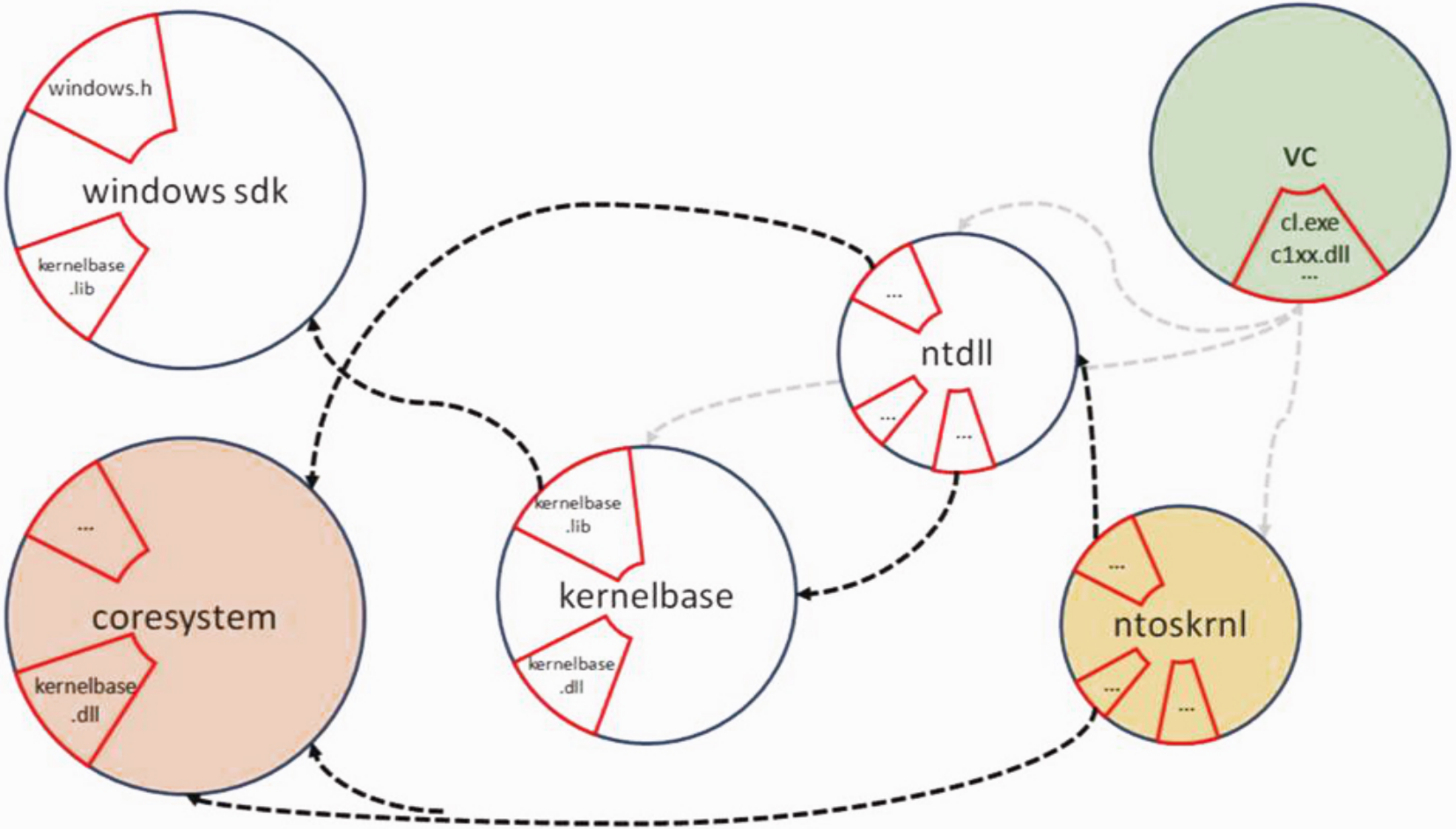
- The *north star* for our first Apex system includes:
  - a source **componentization** system (Airstream)
  - a new **source** code management strategy (Git on VSO)
  - a new **quality** strategy (inner loop)
  - a new **build** language and system (Domino)

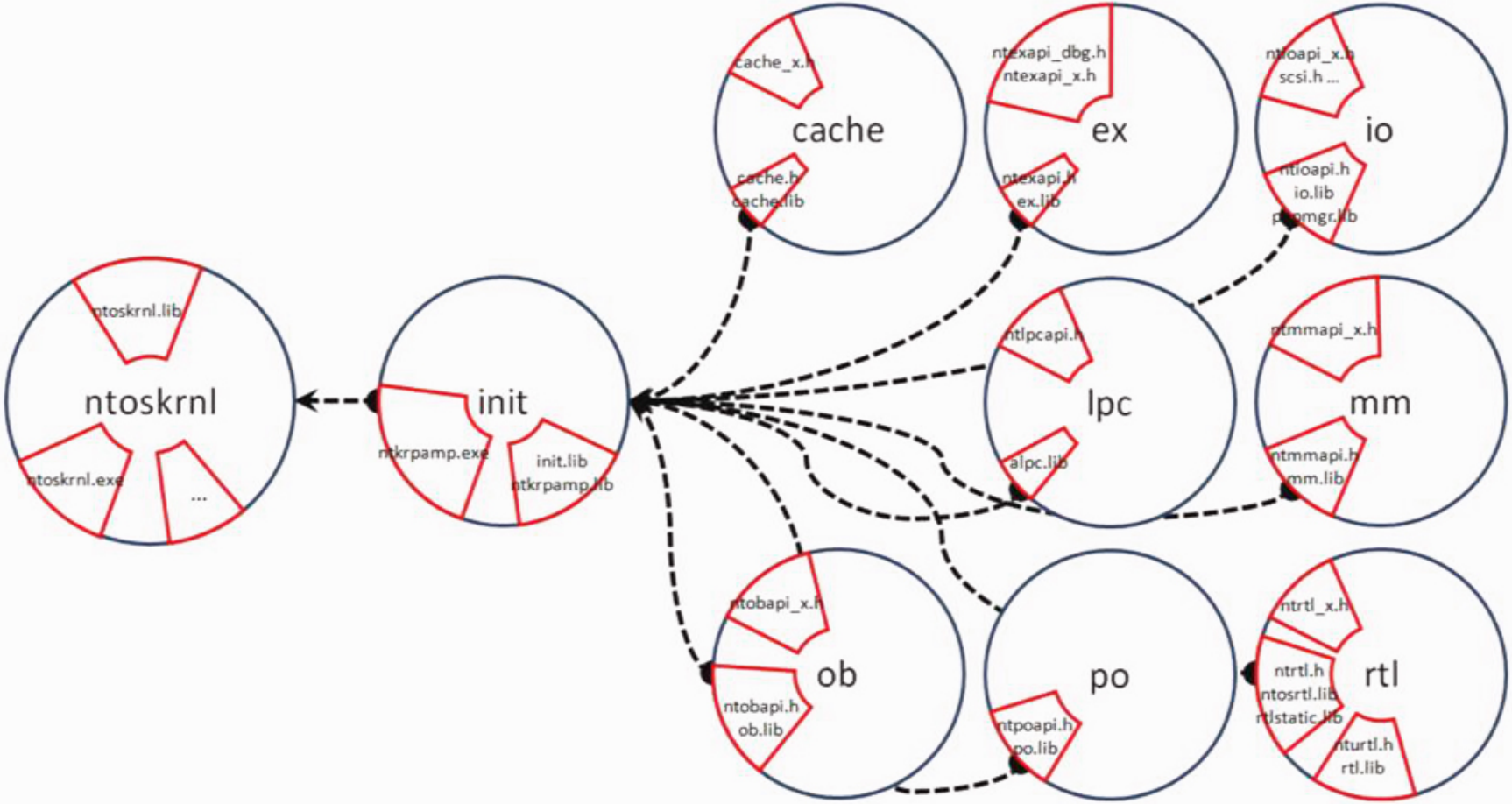
- EDN1 was a straw man to get that going...
- EDN0 describes the process will we use to change it...

Why a source componentization system?

- We've been working on componentization of Windows since 2004… slow progress **because it isn't real for developers**.
- ***Airstream*** allows a first-class, source-based component model:
  - *module* has machine-readable descriptions of:
    - …where to find the sources
    - …how to run the tests
    - …how to package the binaries for deployment
    - …how to associate analytics data with it
  - Consumed through *facets*
- Attributes of good components
  - **Dependencies understood, contracts and usage is explicit**
  - Well layered & aggregate-able
  - **Serviceable & shippable on its own cadence**
  - **Isolated and re-usable (across MS)**



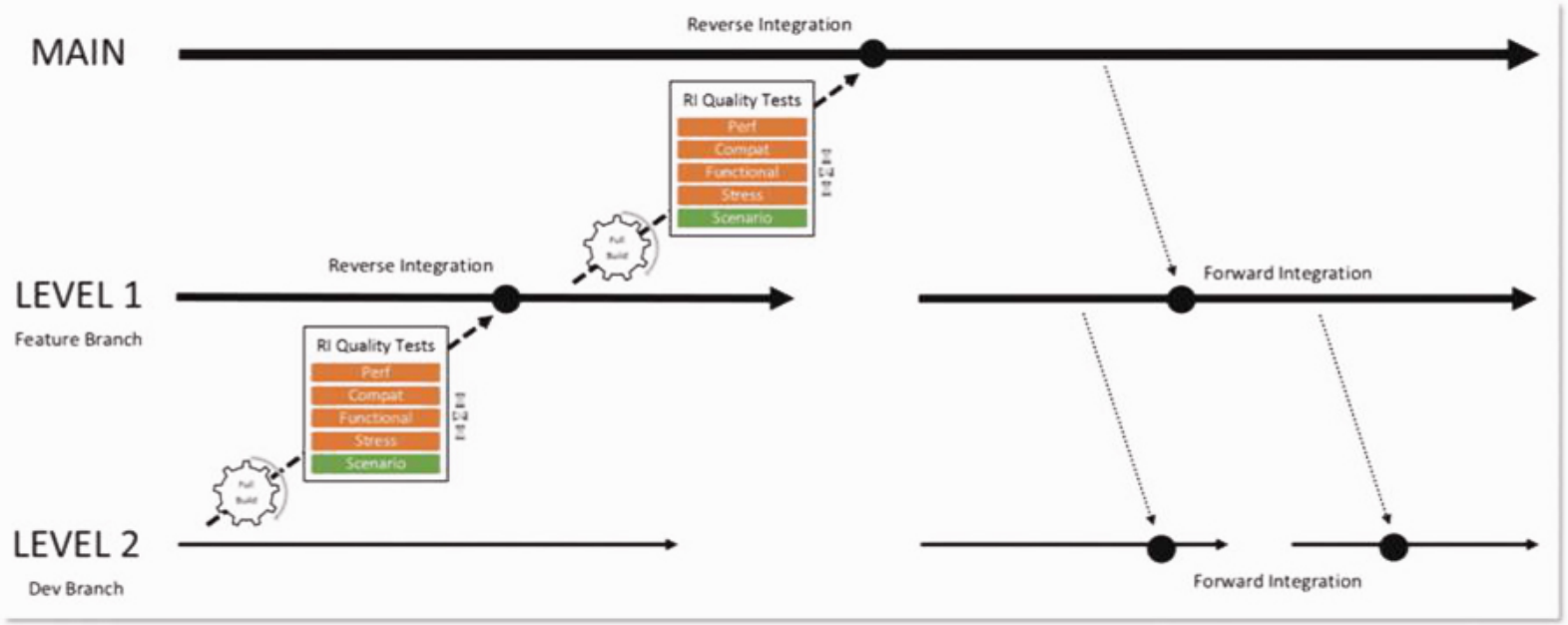

windows.h
windows sdk
kernelbase
.lib
VC
cl.exe
c1xx.dll
...
ntdll
...
...
...
kernelbase
.lib
kernelbase
kernelbase
.dll
...
coresystem
kernelbase
.dll
...
ntoskrnl
...
...


ntoskrnl.lib
ntoskrnl
ntoskrnl.exe
...
init
ntkrpamp.exe
init.lib
ntkrpamp.lib
cache_x.h
cache
cache.h
cache.lib
ntexapi_dbg.h
ntexapi_x.h
ex
ntexapi.h
ex.lib
ntioapi_x.h
scsi.h ...
io
ntioapi.h
io.lib
ntlpcapi.h
lpc
alpc.lib
ntmmapi_x.h
mm
ntmmapi.h
mm.lib
ntobapi_x.h
ob
ntobapi.h
ob.lib
po
ntpoapi.h
po.lib
ntrtl_x.h
ntrtl.h
ntosrtl.lib
rtlstatic.lib
rtl
nturtl.h
rtl.lib

Semantic Versions (Major.Minor.Patch)

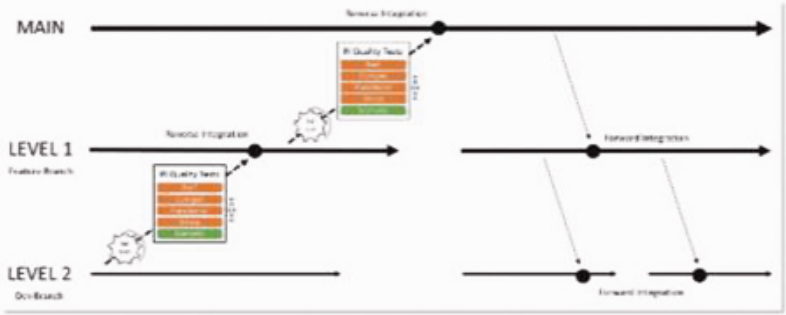
Given a version number 6.5.34

Why a new source code management strategy?



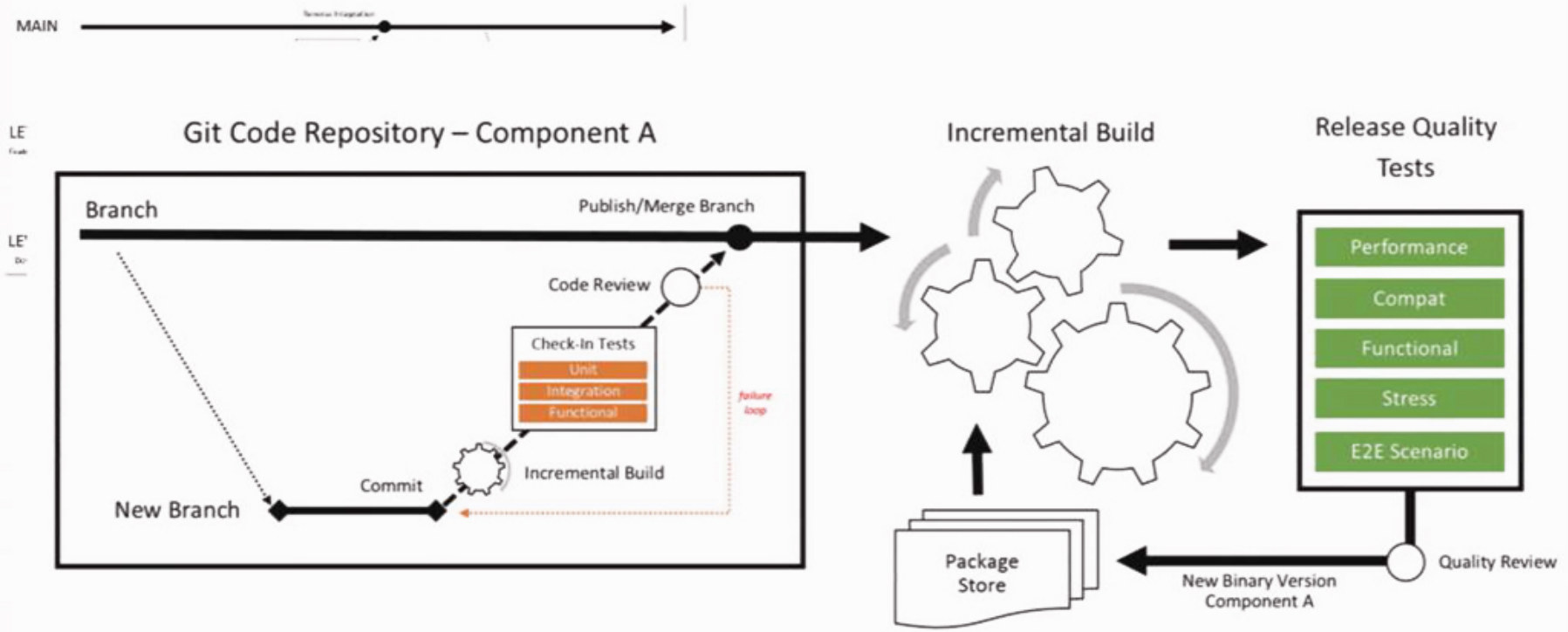
(See EDN3,EDN29)

Why a new source code management strategy?

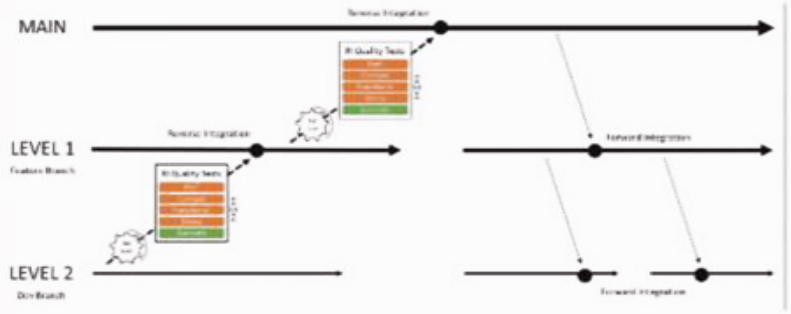


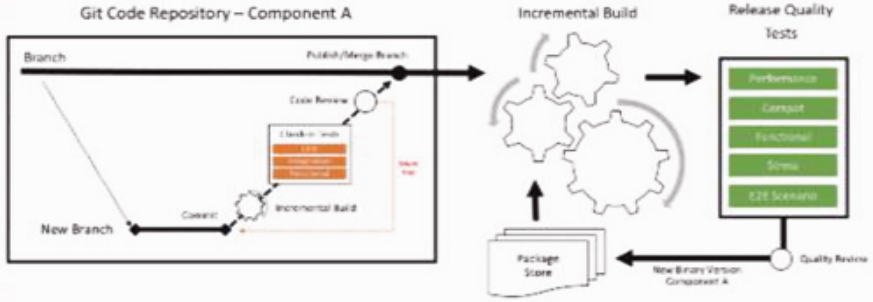
(See EDN3,EDN29)

Why a new source code management strategy?



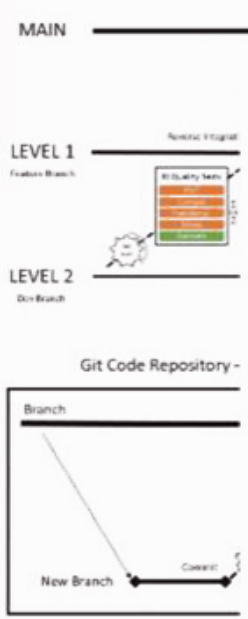
Why a new source code management strategy?





(See EDN3,EDN29)

Why a new source code management strategy?



Code Flow Validation Today

Pre Check-in
- Unit Tests
- Buddy Testing
- Code Review

Post Check-in
- Incremental Build
- SmartQ/TECOV tests

Full Build

Gates & Integration Testing
- TVS, BVTs, Perf, Appcompat, etc...
- Integration tests

RI to next level branch

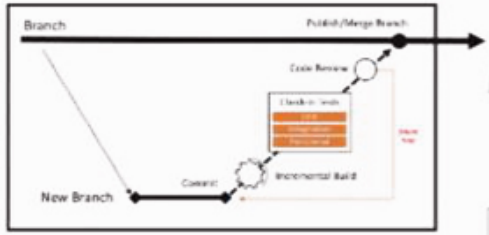
- No standards for pre or post check-in testing
- Full monolithic build

- RI validation repeated at every branch (L3, L2, L1, Main)
- <Insert avg change time data here>
- **Components not in APEX continue to use this system**

N3,EDN29)

Why a new source code management strategy?

Git Code Repository – Component A



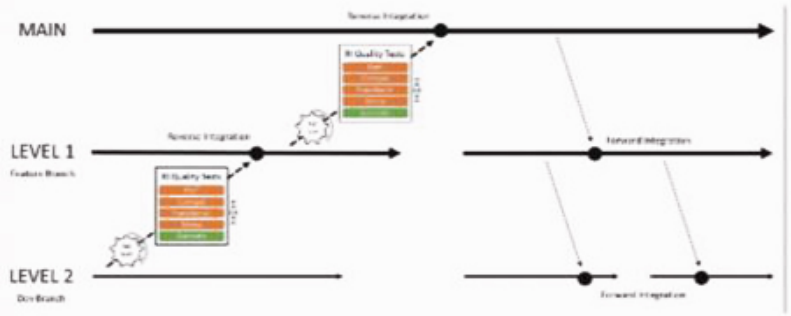
Code Flow Validation Today

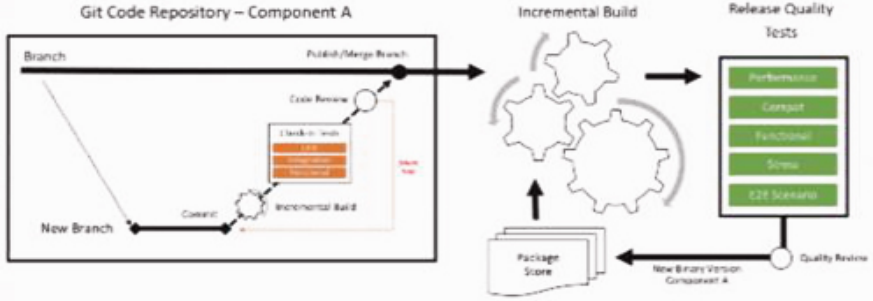
- No standards for pre or post check-in testing
- Full monolithic build

- RI validation repeated at every branch (L3, L2, L1, Main)
- <Insert avg change time data here>
- **Components not in APEX continue to use this system**

(See EDN3,EDN29)

Why a new source code management strategy?



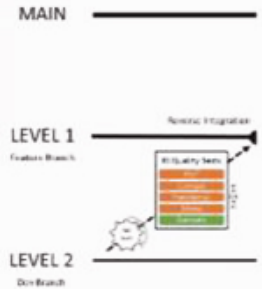


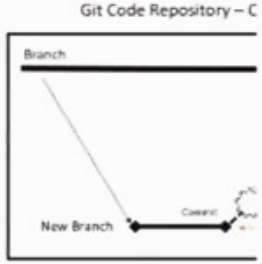
Code Flow Validation Today

- No standards for pre or post check-in testing
- Full monolithic build
- RI validation repeated at every branch (L3, L2, L1, Main)
- <Insert avg change time data here>
- **Components not in APEX continue to use this system**

(See EDN3,EDN29)

Why a new source code management strategy?





Apex Code Flow Validation

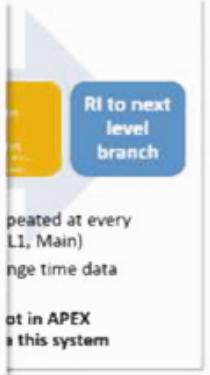
Incremental Build

Inner Loop Tests
- DEV selected
- Extensible Gated Check-in Policy

Code & Results Review

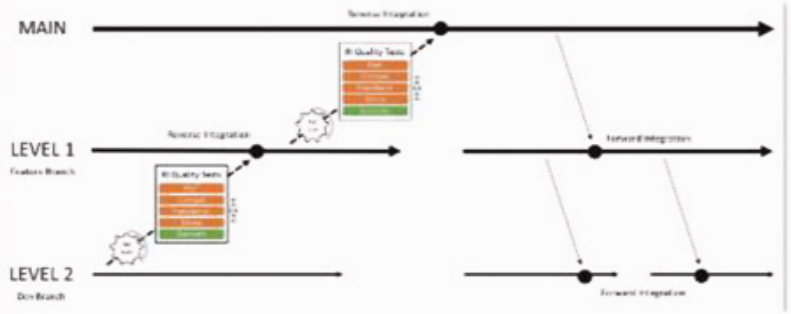
Push
- Change available in .LB fee

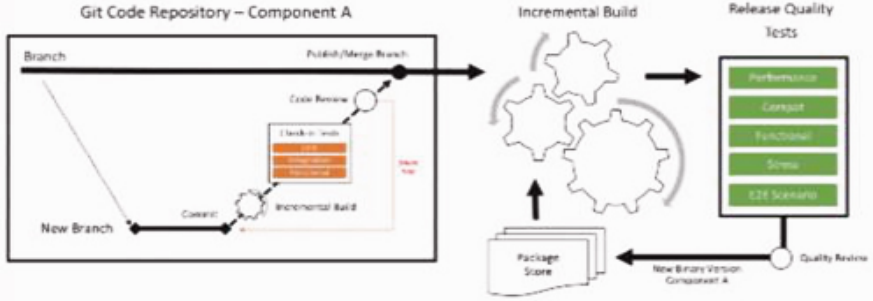
- Check-in validation occurs on incremental build
- Developer is responsible to pick the right unit, component, integration, and/or health tests to ensure self host quality into main
- Changes rapidly available to dependent teams subscribed to .LB feed
- Policies can be defined at team, group, or org level



(See EDN3,EDN29)

Why a new source code management strategy?





Code Flow Validation Today

- No standards for pre or post check-in testing
- Full monolithic build
- RI validation repeated at every branch (L3, L2, L1, Main)
- <Insert avg change time data here>
- **Components not in APEX**

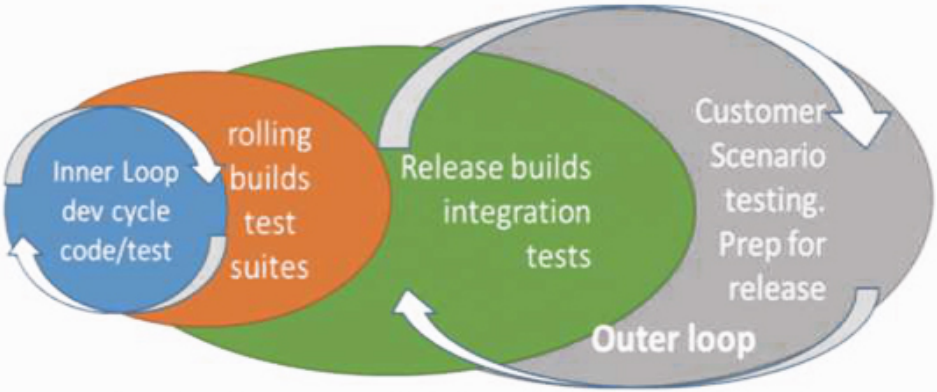
Apex Code Flow Validation

- Check-in validation occurs on incremental build
- Developer is responsible to pick the right unit, component, integration, and/or health tests to ensure self host quality into main
- Changes rapidly available to dependent teams subscribed to .LB feed
- Policies can be defined at team, group, or org level

(See EDN3,EDN29)

Why a new quality strategy?

... to shorten and simplify path from dev to deployment



A high-level overview of the developer workflow. Each loop and test collateral has associated quality and confidence metrics

Inner Loop tests

1) Unit Tests
2) Functional API Tests
3) Component Tests

Properties

1) Quick to run
2) Simple to setup and tear down

(see EDN17)

Why a new build language (Domino)?

- Move from "*just enough project dependencies to build*" to "*devs understand cost of build and full component dependencies*"
- Reliably rebuild only what I changed
- Have a build system that can **scale** to build full Windows RT SKU.
- Ship the build system in Visual Studio.

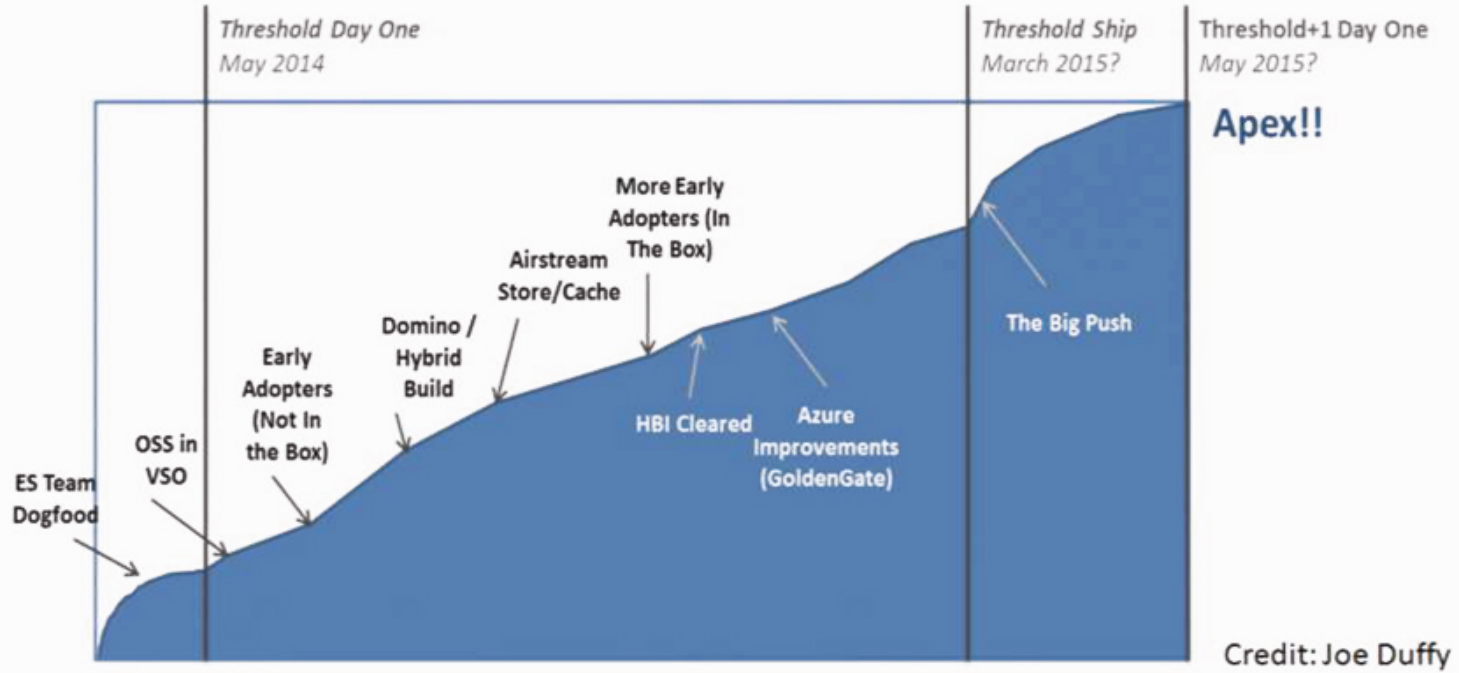
(See EDN13)

Agenda

- Motivation: Why a new Engineering System?
- Direction: What will Apex look like?
- **Call to Action:** What should you do now?

What does the roadmap look like?

… we don't know yet.



Credit: Joe Duffy

What can/should you do now?

- Be part of the **cultural change**:
  - Looking for already built solutions within and outside of Microsoft.
  - Prioritizing joining work efforts instead of building yet-another solution.
- Think (and act) to make it **easier** for people **to use your code**.
- Think (and act) to make it **easier** for you **to use other's code**.

What can/should you do now?

- Reduce your dependencies!
  - Are you using **headers** or **libraries** you don't need?
  - Are you using **APIs** with better lower-level alternatives?
  - Are you using build **tools** gratuitously?
- Carve out your component!
  - What are the **core** parts of your component?
  - Could they have **fewer dependencies** if carved out?
  - Can you **isolate** your headers?
- Prepare for the Inner Loop!
  - Use **TAEF** and ATLAS to write tests
  - Identify tests that need to run in the **inner loop**
  - Identify the tests that need to be stopped or run less frequently in the **outer loop**

Resources

- One Engineering System site: http://engsys
- Engineering System Architecture Discussions: engsysarch DL
- Engineering Design Notes: http://engsys [Design Notes] (EDN0)
- Points of Contact: http://engsys [Points of Contact]
- Engineering Design Note Moderator: mailto:engsysdnm
- Engineering Design Note News: engsysnotes DL